PATENT ABSTRACTS OF JAPAN

(11)Publication number : 08-314752
(43)Date of publication of application : 29.11.1996

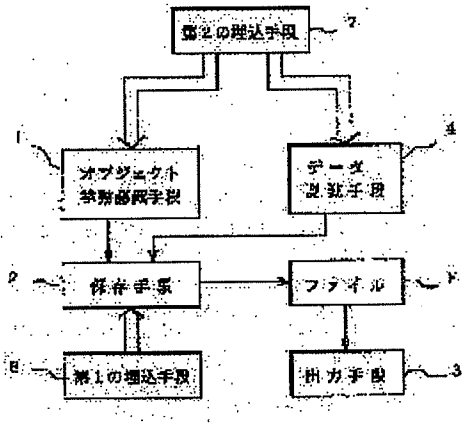
(51)Int.Cl. G06F 11/28
G06F 9/06
G06F 9/44

(21)Application number : 07-142660
(22)Date of filing : 17.05.1995

(71)Applicant : RICOH CO LTD
(72)Inventor : KATO HIDEKAZU

(54) DEVELOPMENT SUPPORTING DEVICE FOR OBJECT-ORIENTED SYSTEM

(57)Abstract:
PURPOSE: To make an user possible to easily understand the behavior of an object-oriented system i.e., the action of an object, etc., when the system is actually operated.
CONSTITUTION: This device is equipped with an object behavior recognizing means 1 which recognizes the transfer (transmission/reception) of a message between the objects, the generation and deletion of the object and the action (behavior) of the object, a preservation means 2 which preserves the behavior of the object recognized by the object behavior recognizing means 1 in a file F and an output means 3 which reads and outputs the behavior of the object preserved in the preservation means.



LEGAL STATUS

[Date of request for examination] 20.02.2001
[Date of sending the examiner's decision of rejection] 23.01.2003
[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]
[Date of final disposal for application]
[Patent number]
[Date of registration]
[Number of appeal against examiner's decision of rejection]
[Date of requesting appeal against examiner's decision of rejection]
[Date of extinction of right]

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号

特開平8－314752

(43)公開日　平成８年(1996)11月29日

(51)Int.Cl.[6]	識別記号	庁内整理番号	ＦＩ		技術表示箇所
Ｇ０６Ｆ　11/28		7313－5B	Ｇ０６Ｆ　11/28	Ａ	
9/06	５４０		9/06	５４０Ｕ	
9/44	５３０	7737－5B	9/44	５３０Ｐ	

審査請求　未請求　請求項の数９　ＦＤ　（全 19 頁）

(21)出願番号　特願平7－142660

(22)出願日　平成７年(1995)５月17日

(71)出願人　000006747
株式会社リコー
東京都大田区中馬込１丁目３番６号

(72)発明者　加藤　英一
東京都大田区中馬込１丁目３番６号　株式会社リコー内

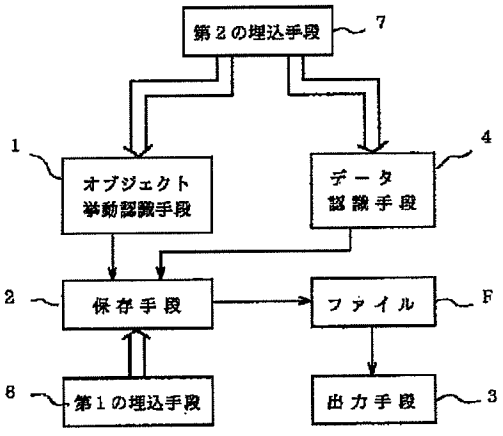
(74)代理人　弁理士　植本　雅治

(54)【発明の名称】　オブジェクト指向システムの開発支援装置

(57)【要約】

【目的】　オブジェクト指向システムを実際に動作させたときのシステムの挙動、すなわち、オブジェクトの振る舞いなどをユーザが容易に理解することが可能である。

【構成】　この開発支援装置は、オブジェクト指向システムの開発環境において、オブジェクト指向システムを実際に起動した際に、オブジェクト間でのメッセージの授受(やりとり)やオブジェクトの生成や削除などのオブジェクトの振る舞い(挙動)を認識するオブジェクト挙動認識手段１と、オブジェクト挙動認識手段１で認識されたオブジェクトの振る舞いをファイルＦに保存する保存手段２と、保存手段によってファイルＦに保存されたオブジェクトの挙動を読み取り出力する出力手段３とを備えている。



1

【特許請求の範囲】

【請求項１】　オブジェクト指向システムの開発環境において、オブジェクト指向システムを実際に起動した際に、オブジェクトの振る舞いを認識するオブジェクト挙動認識手段と、オブジェクト挙動認識手段によって認識されたオブジェクトの振る舞いをファイルに保存する保存手段と、前記保存手段によってファイルに保存されたオブジェクトの振る舞いを読み取り出力する出力手段とを備えていることを特徴とするオブジェクト指向システムの開発支援装置。

【請求項２】　請求項１記載のオブジェクト指向システムの開発支援装置において、前記オブジェクト挙動認識手段によりオブジェクトの振る舞いとしてオブジェクト間でのメッセージの授受が認識されると、前記保存手段は、認識されたメッセージを、オブジェクト間で授受がなされた順にファイルに保存し、前記出力手段は、ファイル保存されたメッセージをも出力するようになっていることを特徴とするオブジェクト指向システムの開発支援装置。

【請求項３】　請求項１記載のオブジェクト指向システムの開発支援装置において、前記オブジェクト挙動認識手段は、特定のオブジェクトに注目し、該特定のオブジェクトに関する振る舞いのみを認識することを特徴とするオブジェクト指向システムの開発支援装置。

【請求項４】　請求項１記載のオブジェクト指向システムの開発支援装置において、さらに、メッセージ間で授受されるメッセージ内のデータを認識するデータ認識手段を有し、前記保存手段は、さらに、データ認識手段で認識されたメッセージ内のデータをもファイルに保存し、前記出力手段は、さらに、ファイルに保存されたメッセージ内のデータをも出力するようになっていることを特徴とするオブジェクト指向システムの開発支援装置。

【請求項５】　請求項１乃至請求項４のいずれか一項に記載のオブジェクト指向システムの開発支援装置において、さらに、前記保存手段をオブジェクトの指向システムの実行前にファイルに埋め込む第１の埋込手段を有していることを特徴とするオブジェクト指向システムの開発支援装置。

【請求項６】　請求項１乃至請求項４のいずれか一項に記載のオブジェクト指向システムの開発支援装置において、さらに、前記オブジェクト挙動認識手段，前記データ認識手段を、オブジェクト指向システムの実行前にファイルに埋め込む第２の埋込手段を有していることを特徴とするオブジェクト指向システムの開発支援装置。

【請求項７】　請求項１乃至請求項４のいずれか一項に記載のオブジェクト指向システムの開発支援装置において、前記出力手段は、前記ファイルに保存された内容を視覚化して表示することを特徴とするオブジェクト指向システムの開発支援装置。

2

【請求項８】　請求項７記載のオブジェクト指向システムの開発支援装置において、前記出力手段は、前記ファイルに保存された内容を視覚化して表示する際に、ユーザの要求に応じて、システムの稼働に合わせて、逐次、システムの稼働状況を表示することを特徴とするオブジェクト指向システムの開発支援装置。

【請求項９】　請求項７記載のオブジェクト指向システムの開発支援装置において、前記出力手段は、オブジェクトの振る舞いをシステムの実行と同時に視覚化して表示することを特徴とするオブジェクト指向システムの開発支援装置。

【発明の詳細な説明】

【０００１】

【産業上の利用分野】本発明は、オブジェクト指向技術を用いるオブジェクト指向システムの開発支援装置に関する。

【０００２】

【従来の技術】従来、オブジェクト指向システムを開発するのに、システム開発に必要なオブジェクトモデル図，事象トレース図，データフロー図などを構築するツールがあった。また、システムの実装を支援するツールとしてはオブジェクトモデル図からクラスの定義をファイルに書き出すようなツールがあった。

【０００３】しかしながら、オブジェクトモデル図，事象トレース図，データフロー図などを構築するツールは、システムの実装に入る前のフェーズ、すなわち分析，設計を支援するためのツールであり、また、オブジェクトモデル図からクラスの定義をファイルに書き出すようなツールは、静的なデータやシステムの実行前の開発段階を支援するものであり、いずれのツールも、システム開発のためのデバッグ，すなわちオブジェクト指向システムを実際に動作させたときのオブジェクト間の時間的流れのデバッグ(動的チェック)を行なうようになっていない。

【０００４】また、従来では、オブジェクト指向システムを開発するのに有効なデバッグツールは、存在していない。これは、オブジェクト指向システムでは、従来のウォーターフォールのように処理が一方向に流れるのとは異なって、オブジェクト間のメッセージのやりとりにより処理が進行し、処理が再び元に戻ることも可能となっているので、現在の処理がどこで行われているか、どのようなデータを扱っているかが見えにくく、また、処理がどのように進行していくのかが把握しにくく、全体としてデバッグ作業が困難であること、また、オブジェクト指向システムでは、オブジェクト指向に特有なクラス階層を用いることにより、継承機能を用いて、差分によるプログラミングを行うことができるが、その反面、上位のクラスに記述されていることは下位のクラスで記述する必要がないため、下位のクラス定義だけを見ただけでは、そのクラスの持つすべての属性やメソッドを把

握したことにはならないことなどに起因している。このため例えば、Ｃ＋＋言語のデバッグをＣ言語のデバッグ環境を流用し行なうというような形で、オブジェクト指向システムのデバッグを行なうための技術は、現在のところ確立されていない。

【０００５】

【発明が解決しようとする課題】本発明は、オブジェクト指向システムを実際に動作させたときのシステムの挙動、すなわち、オブジェクトの振る舞いなどをユーザが容易に理解することの可能なオブジェクト指向システムの開発支援装置を提供することを目的としている。

【０００６】

【課題を解決するための手段および作用】上記目的を達成するために、請求項１乃至請求項９記載の発明では、オブジェクト指向システムの開発環境において、オブジェクト指向システムを実際に起動した際に、オブジェクト間でのメッセージの授受やオブジェクトの生成や削除などのオブジェクトの振る舞いを認識するオブジェクト挙動認識手段と、オブジェクト挙動認識手段によって認識されたオブジェクトの振る舞いをファイルに保存する保存手段と、前記保存手段によってファイルに保存されたオブジェクトの振る舞いを読み取り出力する出力手段とを備えている。これにより、オブジェクト指向システムを実際に動作させたときのシステムの挙動、すなわち、オブジェクトの振る舞いなどをユーザが容易に理解することができる。

【０００７】特に、請求項３記載の発明では、前記オブジェクト挙動認識手段は、特定のオブジェクトに注目し、該特定のオブジェクトに関する振る舞いのみを認識するので、必要なデータのみを集中して取り扱うことができ、不必要なデータを省くことができる。

【０００８】また、請求項６記載の発明では、さらに、前記オブジェクト挙動認識手段，前記データ認識手段を、オブジェクト指向システムの実行前にファイルに埋め込む第２の埋込手段を有しており、第２の埋込手段は、ソースプログラムを解析することによって、オブジェクトの生成や削除，メッセージの送信などの振る舞いの箇所を自動認識してオブジェクト挙動認識手段を埋め込むことができる。これにより、ユーザは、オブジェクト挙動認識手段をこのオブジェクト指向システムに自ら設定する必要がなく、ユーザの作業効率を著しく向上させることができる。

【０００９】また、請求項７記載の発明では、前記出力手段は、前記ファイルに保存された内容を視覚化して表示するようになっており、例えば、システムの振る舞いをユーザに提示する場合には、オブジェクトを四角や円などで表示したり、また、メッセージの授受(やりとり)をユーザに提示する場合には、オブジェクト間にリンクを表示するなどによって、ユーザは視覚的に容易にシステムの振る舞いを把握できる。

【００１０】また、請求項８記載の発明では、前記出力手段は、前記ファイルに保存された内容を視覚化して表示する際に、ユーザの要求に応じて、システムの稼働に合わせて、逐次、システムの稼働状況を表示する。これにより、時間の経過によるシステムの振る舞いをユーザに提示したり、また、時間をさかのぼってシステムの振る舞いをユーザに提示すること(すなわち、時間をさかのぼってシステムの振る舞いをたどること)も可能となる。

【００１１】また、請求項９記載の発明では、オブジェクトの振る舞いをシステムの実行と同時に視覚化して表示する。これにより、ユーザは、リアルタイムに容易にシステムの振る舞いを理解することができる。

【００１２】

【実施例】以下、本発明の実施例を図面に基づいて説明する。図１は本発明に係るオブジェクト指向システムの開発支援装置の一実施例の構成図である。図１を参照すると、この開発支援装置は、オブジェクト指向システムの開発環境において、オブジェクト指向システムを実際に起動した際に、オブジェクト間でのメッセージの授受(やりとり)やオブジェクトの生成や削除などのオブジェクトの振る舞い(挙動)を認識するオブジェクト挙動認識手段１と、オブジェクト挙動認識手段１で認識されたオブジェクトの振る舞いをファイルＦに保存する保存手段２と、保存手段によってファイルＦに保存されたオブジェクトの挙動を読み取り出力する出力手段３とを備えている。

【００１３】ここで、オブジェクト挙動認識手段１によってオブジェクト間でのメッセージの授受が認識されると、保存手段２は、認識されたメッセージを、オブジェクト間で授受がなされた順にファイルに保存し、出力手段３は、ファイル保存されたメッセージをも出力するようになっている。

【００１４】また、オブジェクト挙動認識手段１は、特定のオブジェクトに注目し、該特定のオブジェクトに関する振る舞いのみを認識するよう、これを構成することができる。注目する特定のオブジェクトを指定することを可能にするよう構成する場合には、必要なデータのみを集中して取り扱うことができる。

【００１５】また、この開発支援装置は、さらに、メッセージ間で授受されるメッセージ内のデータを認識するデータ認識手段４を有し、保存手段２は、さらに、データ認識手段で認識されたメッセージ内のデータをもファイルＦに保存し、出力手段３は、さらに、ファイルＦに保存されたメッセージ内のデータをも出力するようになっている。

【００１６】さらに、本実施例の開発支援装置は、保存手段２をオブジェクト指向システムの実行前に、オブジェクト指向システムの所定の箇所に埋め込む第１の埋込手段６と、オブジェクト挙動認識手段１および／または

データ認識手段４を、オブジェクト指向システムの実行前に、オブジェクト指向システムの所定の箇所に埋め込む第２の埋込手段７とを有している。

【００１７】すなわち、システム実行時のオブジェクトの挙動(振る舞い)として、オブジェクト間のメッセージの授受，オブジェクトの生成，削除などを認識する機能を有するオブジェクト挙動認識手段１は、第２の埋込手段７によってオブジェクトの所定の振る舞いの箇所に埋め込まれて使用されるようになっている。また、メッセージ間でやりとりされるメッセージ内のデータを認識する機能を有するデータ認識手段４も、第２の埋込手段７によってオブジェクト間でメッセージの授受がなされる箇所に埋め込まれて使用されるようになっている。

【００１８】より詳細には、上記第１および第２の埋込手段６，７は、ソースプログラムを解析することにより、オブジェクトの生成や削除，メッセージの送信などのオブジェクトの振る舞いの箇所を自動認識して、データ認識手段４，オブジェクト挙動認識手段１を埋め込むようになっており、ユーザがオブジェクトの振る舞いの箇所を教える必要がなく、これらを自動的に埋め込む機能を備えている。

【００１９】また、上記出力手段３は、ファイルＦに保存された内容を視覚化して例えば表示するようになっている。より詳しくは、出力手段３は、例えば、オブジェクトの振る舞いが書き出されたファイルＦを読み込み、システムの振る舞いをユーザに例えば表示するようになっている。例えば、オブジェクトを四角や円などで、またメッセージのやりとりの際にはオブジェクト間にリンクを表示するなどして、システムの振る舞いをユーザが視覚的に容易に把握できる形に出力するようになっている。

【００２０】なお、出力手段３としては、ファイルＦに保存された内容を視覚化して例えば表示する際に、さらに、ユーザの要求に応じてシステムの稼働に合わせて、逐次、システムの稼働状況をも表示する機能を有していても良い。これにより、時間の経過によるシステムの振る舞いをユーザに提示したり、また、時間をさかのぼってシステムの振る舞いを逆にたどることも可能となる。

【００２１】また、出力手段３としては、オブジェクトの振る舞いをシステムの実行と同時に視覚化して表示する機能を有していても良い。デバッグ中にデバッグと同時に視覚化してユーザにオブジェクトの振る舞いを表示する機能を出力手段３が備えている場合には、デバッグ中のソースプログラムと視覚化されたオブジェクトの振る舞いとを同時進行させて表示させることが可能となる。

【００２２】次に、本実施例の開発支援装置の処理動作例について説明する。図２は、本実施例の開発支援装置の処理動作の一例を示すフローチャートである。

【００２３】図２の処理動作例では、先ず、ステップＳ１において、ソースプログラムを指定し、開発システムのファイル構成をチェックする。これにより、この開発支援装置が次のステップＳ２以降で行うソースプログラムの解析の範囲を限定する。

【００２４】次に、ステップＳ２では、ソースプログラムを解析する。具体的に、このステップＳ２では、ソースプログラムの解析処理として、ユーザが定義するクラスやデータ型などの認識処理を行なう。すなわち、開発支援環境がオブジェクトの振る舞いを理解するために、クラスの定義の認識処理を行ない、クラスがどのような属性およびメソッドを持つかを知得する。また、オブジェクト間でのメッセージのやり取り(メソッドの呼び出し)の状況も理解する必要があるために、クラスの属性やメソッドの返り値のデータ型、およびメソッドの持つ引数の型でユーザ定義の型も認識する。

【００２５】また、ステップＳ２の解析処理では、プログラム中のオブジェクトの生成，削除やメソッドの呼び出しおよびその他のオブジェクトの振る舞いを認識する処理を行なう。また、これと同時に、第１および第２の埋込手段６，７によって所定の振る舞いの箇所にオブジェクトの振る舞いを書き出す手続きを埋め込む処理を行なう。この手続きを行なうことにより、ステップＳ３でのシステムの実行時に、オブジェクトの生成，削除，メッセージのやり取りなどの振る舞いをファイルに保存させ、また、この時の動的なデータの値も同時にファイルに書き出させることが可能になる。

【００２６】このようにして、ステップＳ１，Ｓ２において、システムを実行するための前処理を行ない、しかる後、ステップＳ３でシステムを実行する。システムを実行するとオブジェクトが種々の振る舞いをするが、この振る舞いの履歴は、ステップＳ２で埋め込んだオブジェクトの振る舞いをファイルに書き出す手続きにより、すなわちオブジェクト挙動認識手段１(さらにはデータ認識手段４)，保存手段２によって、ファイルに書き出され、オブジェクトの振る舞いに関する実データが時系列にファイルに保存される。

【００２７】このようにしてシステムを実行した後、ステップＳ４では、出力手段３を起動し、ステップＳ３でオブジェクトの振る舞いの履歴が保存されたファイルから内容を読み込み、ユーザとの対話を行いながらオブジェクトの振る舞いを逐次にまたは一括に例えば表示する。

【００２８】このように、本実施例の開発支援装置では、オブジェクト指向システムを実際に動作させたときのシステムの挙動、すなわち、オブジェクトの振る舞いなどをユーザが容易に理解することができ、これにより、オブジェクト指向システムを実際に動作させたときのオブジェクト間の時間的流れのデバッグ(動的チェック)を行なうことができて、デバッグ作業を容易に効率良く行なわせることが可能となる。

【００２９】なお、図２の処理例では、システム実行後、オブジェクトの振る舞いを出力(表示)するようになっているが、システムの実行と共にリアルタイムにオブジェクトの振る舞いを出力(表示)することも可能である。

【００３０】図３は、オブジェクトの振る舞いをシステムの実行と同様にリアルタイムに表示する処理例を示すフローチャートである。図３の処理例において、ステップＳ１１，Ｓ１２では、図２のステップＳ１，Ｓ２と同様に、ソースプログラムを指定し、そのプログラムを解析してユーザの定義したクラス定義および型を認識し、オブジェクトの振る舞いを示す手続きを埋め込む。

【００３１】次に、ステップＳ１３でシステムを起動するが、この際、図２の処理例と異なり、図３の処理例では、システムの実行とともにオブジェクトの振る舞いを表示するために、この時点で出力手段３を起動する。そして、システムの実行に伴ないオブジェクトの振る舞いを示す手続き，すなわちオブジェクト挙動認識手段１(さらにはデータ認識手段４)，保存手段２が起動され、オブジェクトの振る舞いが出力手段３によって逐次表示される。

【００３２】なお、図２のステップＳ２では、オブジェクトの振る舞いを示す手続きを埋め込む際に、ファイルにオブジェクトの振る舞いを保存していたが、図３のステップＳ１２では、リアルタイムにオブジェクトの振る舞いを表示する必要があることから、このシステム内にオブジェクトの振る舞いを管理するクラスまたは機能を設け、このクラスにオブジェクトの振る舞いを通知する手続きを用いることもできる。もちろん、ステップＳ１２においても、図２のステップＳ２と同様に、ファイルにオブジェクトの振る舞いを保存しておき、別のプロセスで出力手段３を起動しておいて、その出力手段３にそのファイルを随時チェックさせて、オブジェクトの振る舞いを表示させてもよい。さらにまた、プロセス間通信などの手法を用いてオブジェクトの振る舞いを別プロセスである開発支援装置に通知することにより実現することもできる。

【００３３】また、図２，図３の処理例において、ステップＳ２，Ｓ１２でのソースプログラムの解析処理は、具体的に以下のようにして行なうことができる。

【００３４】すなわち、ソースプログラムの解析は、上述したように、各定義の認識と、各オブジェクトの振る舞いの認識との２つの処理に大きく分けられる。各定義を認識する理由としては、オブジェクトはユーザ(ソースプログラムの創作者)が定義したクラスに基づいているので、オブジェクトの振る舞いを認識するためには、先ず、定義されているクラスをチェックする必要がある。すなわち、クラスがどのような属性やメソッドをもつかを知る必要がある。また、クラスのもつ属性やメソッドの引き数や返り値などに、システムが定義している以外のユーザが定義した新たなデータ型を使用していることもあるので、プログラム内で使用されている全ての変数、および全てのデータ型をチェックする。

【００３５】各定義を認識する手順は、データ型表の作成手順と、クラス表の作成手順と、変数表の作成手順とに、大別できる。これらの手順の後に、オブジェクトの振る舞いの認識を行なわせるため、予め用意された振る舞いに対応したプログラムコードを所定の箇所に埋め込む。

【００３６】各定義を認識する手順のうち、データ型表の作成手順では、Ｃ＋＋で予め用意されているデータ型に加えて、ユーザが定義しているデータ型をチェックし、図４に示すようなデータ型表を作成する。なお、ユーザが定義しているデータ型は、プログラム中のキーワードを検索し、抽出する。そのキーワードとは、ユーザが型を定義する時に使用するための予約語であるtypedefや、定義を宣言するときに使用される予約語＃defineや、クラスの宣言であるclassなどであり、それらの構文にしたがって、データ型の名称をキーワードとして抽出することができる。

【００３７】また、クラス表の作成手順では、クラス内部の属性やメソッドの抽出を行なう。すなわち、クラスもユーザ定義のデータ型の一種であり、従って、クラスはデータ型表の作成手順においても抽出されるが、クラス表の作成手順では、クラス内部の属性やメソッドの抽出を行なう。すなわち、クラス定義部分の抽出は、前述のように、Ｃ＋＋の予約語であるclassを検索することにより行なわれ、抽出されたクラス定義部分からそのクラスのもつ属性の名称および型，メソッドの名称および返り値の型をチェックし、図５に示すようなクラス表を作成する。

【００３８】また、変数表の作成手順では、ソースプログラムを順にチェックし、使用された変数を、グローバル変数とその型、関数と関数内で宣言されている変数とその型、メソッドとメソッド内で宣言されている変数とその型、に分けて抽出し、図６に示したような変数表を作成する。なお、変数の宣言部分は、データ型表の作成で抽出したデータ型名をプログラム中で検索し、変数宣言の構文に従って抽出することができる。

【００３９】このようにして、データが抽出されると、抽出したデータに基づいて、デバッグのための情報をプログラムに埋め込んでいく。先ず、ソースプログラムを順にチェックし、オブジェクトの振る舞いの箇所を検索し、その箇所にその振る舞いに対応したデバッグ用プログラムコードを予め埋め込んでいく。

【００４０】オブジェクトの振る舞いの代表的なものとしては、オブジェクトの生成と、オブジェクトの削除と、オブジェクトの持つメソッドの呼び出しと、オブジェクトの持つ属性への値の代入とがあり、これらに対応したデバッグ用プログラムコードの埋め込みについて、

説明する。

【００４１】先ず、オブジェクトの生成に対応したデバッグ用プログラムコードの埋め込みについて説明する。Ｃ＋＋では、オブジェクトの生成は、予約語であるｎｅｗを用いて行なわれるので、プログラム中でこの予約語ｎｅｗを検出し、オブジェクトの生成箇所を検索することができる。

【００４２】このようにしてオブジェクトの生成箇所が検索されると、オブジェクトの生成箇所には図７に示すようなデバッグ用プログラムコードを埋め込むことができる。図７のデバッグ用プログラムコードは、オブジェクトの生成の情報として、すべてのオブジェクトの振る舞いに対する通し番号であるｃｏｕｎｔを表示するためのコードと、オブジェクトの生成を示すＣＲＥＡＴＥ，生成されるクラスの名称(クラス名)を表示するためのコードと、生成されたオブジェクトが格納される変数ＶＡＲＩＡＢＬＥの名称(変数名)を表示するためのコードと、オブジェクトがどこで生成されたかを示すＷＨＥＲＥ，およびその生成箇所を表示するためのコードと、生成箇所の名称を表示するためのコードとにより構成されたものとなっている。なお、オブジェクト指向システムでは、オブジェクトの生成箇所が関数内である場合とメソッド内である場合とがあり、従って、生成箇所の名称を表示するためのコードとしては、関数名またはメソッド名が用いられる。

【００４３】次に、オブジェクトの削除に対応したデバッグ用プログラムコードの埋め込みについて説明する。Ｃ＋＋では、オブジェクトの削除は、予約語であるｄｅｌｅｔｅを用いて行なわれるので、プログラム中でこの予約語ｄｅｌｅｔｅを検出し、オブジェクトの削除箇所を検索することができる。

【００４４】このようにしてオブジェクトの削除箇所が検索されると、オブジェクトの削除箇所には、図８に示すようなデバッグ用プログラムコードを埋め込むことができる。図８のデバッグ用プログラムコードは、オブジェクトの削除の情報として、すべてのオブジェクトの振る舞いに対する通し番号であるｃｏｕｎｔを表示するためのコードと、オブジェクトの削除を示すＤＥＬＥＴＥ，削除されるクラスの名称(クラス名)を表示するためのコードと、削除されたオブジェクトが格納されていた変数ＶＡＲＩＡＢＬＥの名称(変数名)を表示するためのコードと、オブジェクトがどこで削除されたかを示すＷＨＥＲＥ，およびその削除箇所を表示するためのコードと、削除箇所の名称を表示するためのコードとにより構成されたものとなっている。なお、オブジェクト指向システムでは、削除箇所が関数内である場合とメソッド内である場合とがあり、従って、削除箇所の名称を表示するためのコードとしては、関数名またはメソッド名が用いられる。

【００４５】次に、オブジェクトの持つメソッドの呼び出しに対応したデバッグ用プログラムコードの埋め込みについて説明する。Ｃ＋＋では、オブジェクトの持つメソッドの呼び出しは、“->”または“.”で表わされるので、プログラム中でこの語を検出し、オブジェクトの持つメソッドの呼び出し箇所を検索することができる。ただし、“->”または“.”のすべてがメソッドの呼び出しとは限らないので、“->”または“.”の左部の変数を変数表にてチェックして、その型がクラス型であることを確認し、また、右部をクラス表でチェックしてメソッドであることを確認する必要がある。

【００４６】オブジェクトの持つメソッドの呼び出し箇所には、図９に示すようなデバッグ用プログラムコードを埋め込むことができる。図９のデバッグ用プログラムコードは、オブジェクトの持つメソッドの呼び出しの情報として、すべてのオブジェクトの振る舞いに対する通し番号であるｃｏｕｎｔを表示するためのコードと、オブジェクトの持つメソッドの呼び出しを示すＭＥＴＨＯＤ，メソッドの名称(メソッド名)を表示するためのコードと、オブジェクトの持つメソッドがどこから呼び出されたかを示すＦＲＯＭ，メソッドの呼び出し箇所を表示するためのコードと、メソッドの呼び出し箇所の名称を表示するためのコードと、どのクラスの持つメソッドが呼び出されたかを示すクラス名(呼び出し先クラス名)を表示するためのコードと、呼び出されたメソッドの引き数の数を表示するためのコードと、呼び出されたメソッドの引き数の値を表示するためのコードとにより構成されたものとなっている。なお、オブジェクト指向システムでは、メソッドの呼び出し箇所が関数内である場合とメソッド内である場合とがあり、従って、メソッドの呼出し箇所の名称を表示するためのコードとしては、関数名またはメソッド名が用いられる。

【００４７】次に、オブジェクトの持つ属性への値の代入に対応したデバッグ用プログラムコードの埋め込みについて説明する。Ｃ＋＋では、オブジェクトの持つ属性は、クラス内での属性は、そのまま名称で、クラス外での属性は、“->”または“.”で表わされるので、プログラム中でこの語を検出し、オブジェクトの持つ属性への値の代入箇所を検索することができる。ただし、“->”または“.”のすべてが値の代入箇所とは限らないので、“->”または“.”の左部の変数を変数表にチェックしてその型がクラス型であることを確認し、また、右部をクラス表でチェックして属性名であることを確認する必要がある。

【００４８】このようにしてオブジェクトの持つ属性への値の代入箇所が検索されると、オブジェクトの持つ属性への値の代入箇所には、図１０に示すようなデバッグ用プログラムコードを埋め込むことができる。図１０のデバッグ用プログラムコードは、オブジェクトの持つ属性への値の代入の情報として、すべてのオブジェクトの振る舞いに対する通し番号であるｃｏｕｎｔを表示する

ためのコードと、オブジェクトの持つ属性への値の代入を示すＡＴＴＲＩＢＵＴＥ，属性の名称(属性名)を表示するためのコードと、値をどこで代入しているかを示すＷＨＥＲＥ，値の代入箇所を表示するためのコードと、値の代入箇所の名称を表示するためのコードと、どのクラスの持つ属性に値が代入されたかを示すクラス名(呼び出し先クラス名)を表示するためのコードと、どんな値が代入されたかを示す変数ＶＡＲＩＡＢＬＥの名称(変数名)を表示するためのコードとにより構成されている。なお、オブジェクト指向システムでは、属性への値の代入箇所が関数内である場合とメソッド内である場合とがあり、従って、属性への値の代入箇所の名称を表示するためのコードとしては、関数名またはメソッド名が用いられる。

【００４９】このようにして、本実施例では、ソースプログラムにおいて、オブジェクトの振る舞いの箇所を自動検索し、検索した箇所に所定のデバッグ用プログラムコードを自動的に埋め込むことができる。なお、このような検索，埋め込み処理を、上述のように、例えば、オブジェクトの生成箇所，オブジェクトの削除箇所，オブジェクトの持つメソッドの呼び出し箇所，オブジェクトの持つ属性への値の代入箇所ごとに、各々別個のルーチンで行なっても良いが、これらの各検索，埋め込み処理を１つのルーチンで１パスで実行することもできる。

【００５０】また、上記例では、ソースプログラム中へのオブジェクト挙動認識手段１の埋め込みについて具体的に説明したが、データ認識手段４，保存手段２のソースプログラム中への埋め込み処理についても、同様にして、自動的に行なうことができる。

【００５１】このようにして、ソースプログラムを解析し、ユーザが定義したクラスの定義(属性およびメソッド)やユーザ定義のデータタイプ(型)などを取り出し、プログラム中のメッセージの呼び出し箇所およびその時のデータとデータタイプを認識し、さらにその他のオブジェクトの振る舞いを自動認識して、その振る舞いを行う箇所に、所定のデバッグ用プログラムコードを自動的に埋め込むことができる。

【００５２】いま、例えば図１１に示すようなソースプログラム(サンプルプログラム)が指定されたとし、このサンプルプログラムを用いてオブジェクトの振る舞いがどのように本実施例の開発支援装置で処理されるかを図２の処理を例にとって具体的に説明する。

【００５３】図２のステップＳ１の処理においては、図１１に示すようなソースプログラムが与えられるとき、これをファイル名がｔｅｓｔ．ｃｃであるものとして指定する。次に、ステップＳ２におけるソースプログラム(ｔｅｓｔ．ｃｃ)の解析処理では、まず、ファイルの先頭に、オブジェクトの履歴を書き出すファイルを宣言しておく。さらに、ソースプログラムからｍａｉｎ関数を検索し、ｍａｉｎ関数の先頭にこの履歴を保存するファイルをオープンする関数を挿入する。また、ｍａｉｎ関数の最後にこのファイルをクローズする関数を挿入する。次にファイルを順次解析していき、ファイル内で定義されているクラスおよびデータ型を認識する。図１１のソースプログラムの場合には、クラス定義として、ｃｌａｓｓＢおよびｃｌａｓｓＡが定義されており、ユーザ定義のデータ型はない。また、クラス定義の中でｃｌａｓｓＢはメソッドｍｅｔｈｏｄＢをもち、また、ｃｌａｓｓＡは属性ｐｔｒとメソッドｍｅｔｈｏｄＡをもち、ｍｅｔｈｏｄＢの引数はｃで、ｃｈａｒであり、ｃｌａｓｓＡの属性ｐｔｒのタイプはｃｌａｓｓＢへのポインターであり、また、ｍｅｔｈｏｄＡの引数はａ，ｂの２つで、ともにｉｎｔｅｇｅｒであることが認識される。またｍａｉｎ関数内で変数ａ，ｂがそれぞれクラスのｃｌａｓｓＡ型，ｃｌａｓｓＢ型であることなどの変数の型も認識される。このようにして、図４，図５，図６に示したようなデータ型表，クラス表，変数表を作成し、これらの情報を基に再びソースプログラムを順次解析していき、オブジェクトが振る舞う箇所を検出し、そこにオブジェクトの振る舞いを書き出す手続きを埋め込む。すなわち、図７，図８，図９あるいは図１０に示すようなデバッグ用プログラムコードを埋め込む。

【００５４】図１１のサンプルプログラムでは、部分Ｐ１，Ｐ２，Ｐ３，Ｐ４，Ｐ５，Ｐ６がオブジェクトの振る舞いの箇所として検出される。

【００５５】すなわち、サンプルプログラム(ｔｅｓｔ．ｃｃ)を上から順に見ていくと、まず、ｃｌａｓｓＡのメソッドｍｅｔｈｏｄＡの中でｍｅｔｈｏｄＢを呼び出しているのが検出される(図１１の部分Ｐ１)。そこで、ここにメソッド呼び出しの情報をファイルに書き出す手続き，すなわち図９に示すようなデバッグ用プログラムコードを埋め込む。

【００５６】次に、ｎｅｗでｃｌａｓｓＡを生成している部分が検出される(図１１の部分Ｐ２)。そこで、ここにはオブジェクト生成の情報をファイルに書き出す手続き，すなわち図７に示すようなデバッグ用プログラムコードを埋め込む。

【００５７】次に、ｃｌａｓｓＢを生成している部分が検出され(図１１の部分Ｐ３)、この部分Ｐ３についても、部分Ｐ２と同様にして、図７に示すようなデバッグ用プログラムコードを埋め込む。

【００５８】次に、プログラム内からのメソッドの呼び出しの部分が検出され(図１１の部分Ｐ４)、この部分Ｐ４についても、先程のｍｅｔｈｏｄＡ内からのｍｅｔｈｏｄＢの呼び出しと同様な情報をファイルに書き出す手続き，すなわち図９に示すようなデバッグ用プログラムコードを埋め込む。

【００５９】そして、最後に、オブジェクトａ、ｂを削除している部分が検出され(図１１の部分Ｐ５，Ｐ６)、これらの部分Ｐ５，Ｐ６にオブジェクト削除の情報をフ

ァイルに書き出す手続き，すなわち図８に示すようなデバッグ用プログラムコードを埋め込む。

【００６０】このようにしてソースプログラム(サンプルプログラム)の解析を終了する。図１２乃至図１４は、図１１のソースプログラムの解析が終了し、図１１のソースプログラムのデバッグに必要な手続き，すなわち所定のデバッグ用プログラムコードが埋め込まれた状態を示す図である。

【００６１】なお、図１２乃至図１４のソースプログラムにおいて、本実施例の開発支援装置によって図１１のソースプログラムに付け加えられた(埋め込まれた)デバッグ用プログラムコードの部分は、すべて、＃ｉｆｄｅｆ　ＯＯ＿ＴＯＯＬと＃ｅｎｄｉｆとで挟まれている。

【００６２】図１２乃至図１４を参照すると、図１２の(Ｑ１)の部分では、オブジェクトの振る舞いを書き出すファイルを定義し、そのためのファイルポインターを宣言している。また、このファイルポインターの宣言に必要なＦＩＬＥ型のために＜ｓｔｄｉｏ．ｈ＞をインクルードしている。ｃｏｕｎｔの変数は、オブジェクトの振る舞いをカウントするためのグローバル変数である。

【００６３】また、図１２の(Ｑ２)の部分(図１１のソースプログラムの部分Ｐ１に対応)には、ｃｌａｓｓＡの持つメソッドｍｅｔｈｏｄＡ内での他のクラスの持つメソッドの呼び出しをファイルに保存する手続き，すなわち図９に示すようなデバッグ用プログラムコードが埋め込まれている。すなわち、図１１のソースプログラムの部分Ｐ１では、ｃｌａｓｓＡのメソッドｍｅｔｈｏｄＡからの呼び出しであるので、図９のデバッグ用プログラムコードにおいて、メソッドの呼び出し箇所は(FROM)、ｃｌａｓｓＡであり、関数名は(FUNCTION)、ｍｅｔｈｏｄＡとなる。

【００６４】これにより、図１２の(Ｑ２)の部分が実行されると、先ず、オブジェクトの振る舞いのカウンターｃｏｕｎｔが出力され、次にメソッドの呼び出しであることを示す”ＭＥＴＨＯＤ”とメソッドの名称ｍｅｔｈｏｄＢが出力される。次に、どこからどこへ(どのクラスの)メソッドが呼ばれたかがＦＲＯＭとＴＯで出力される。なお、これらの間の”ＦＵＮＣＴＩＯＮ”は、どの関数から呼び出されたかを示すものである。この場合は、メソッドｍｅｔｈｏｄＡであるので、この名称ｍｅｔｈｏｄＡが出力される。次いで、引数について、まず、引数の数“１”が出力され、次に引数の値“ａ”が出力される。

【００６５】また、図１３，図１４はｍａｉｎ関数についてのプログラムコードであり、図１３の(Ｑ３)の部分(図１１のソースプログラムの部分Ｐ２に対応)には、プログラムの最初に呼ばれるメイン関数の先頭にオブジェクトの振る舞いを書き出す為のファイルをオープンする手続きが埋め込まれている。また、(Ｑ４)の部分では、オブジェクトの生成を示す内容をファイルに保存する手続き，すなわち、図７に示すようなデバッグ用プログラムコードが埋め込まれている。すなわち、図１１のソースプログラムの部分Ｐ２では、ｍａｉｎという関数内での生成であるので、ともに生成箇所は(WHERE)は、ＦＵＮＣＴＩＯＮであり、関数名は、(FUNCTION)、mainとなる。

【００６６】これにより、(Ｑ４)の部分が実行されると、まずオブジェクトの振る舞いのカウンターｃｏｕｎｔが出力され、次に生成を示す”ＣＲＥＡＴＥ”と生成されるクラスの名称ｃｌａｓｓＡが出力される。次に、生成されたオブジェクトを示すポインターが格納されている変数の名称“ａ”が出力され、次にどこで生成されたかが出力される。いまの場合は、関数内で生成されているので”ＦＵＮＣＴＩＯＮ”と出力されるが、あるクラスの持つメソッド内で生成される場合には、そのクラス名が出力される。次いで、関数の名称またはメソッドの名称が出力される。いまの場合、関数の名称ｍａｉｎが出力される。

【００６７】また、(Ｑ５)の部分(図１１のソースプログラムの部分Ｐ３に対応)にも、(Ｑ４)と同様に、オブジェクトの生成を示す内容をファイルに保存する手続きが埋め込まれており、(Ｑ５)の部分が実行されると、クラスｃｌａｓｓＢに関して、(Ｑ４)の部分と同様の出力がなされる。

【００６８】また、図１３の(Ｑ６)の部分(図１１のソースプログラムの部分Ｐ４に対応)には、図１２の(Ｑ２)とほぼ同様の手続きが埋め込まれている。但し、(Ｑ２)の部分と異なり、(Ｑ６)の部分は、メソッドの呼び出し元がオブジェクトではないので”ＦＲＯＭ”が”ＦＵＮＣＴＩＯＮ”であり、呼び出している関数が”ｍａｉｎ”となる。すなわち、図１１のソースプログラムの部分Ｐ４では、ｍａｉｎという関数内でのメソッドの呼び出しであるので、メソッドの呼び出し箇所は(FROM)、ＦＵＮＣＴＩＯＮであり、関数名は(FUNCTION)、ｍａｉｎとなる。

【００６９】また、図１４の(Ｑ７)の部分(図１１のソースプログラムの部分Ｐ５に対応)には、オブジェクトの削除を示す内容を書き出す手続き，すなわち図８に示すようなデバッグ用プログラムコードが埋め込まれている。(Ｑ７)の部分が実行されると、最初にオブジェクトの振る舞いのカウンターｃｏｕｎｔが出力され、次に削除を示す”ＤＥＬＥＴＥ”および削除されるクラスの名称(ここでは”ｃｌａｓｓＡ”)が出力される。次にこのオブジェクトを示す変数名“ａ”が出力され、次いで、削除される場所”ＷＨＥＲＥ”、ここではクラス内ではなく、関数内なので、”ＦＵＮＣＴＩＯＮ”が出力され、次にその関数名”ｍａｉｎ”が出力される。

【００７０】また、図１４の(Ｑ８)の部分(図１１のソースプログラムの部分Ｐ６に対応)には、(Ｑ７)と同様の手続きが埋め込まれている。

15

【００７１】また、図１４の(Ｑ９)の部分は、図１３の(Ｑ３)のファイルのオープンに呼応する処理で、オブジェクトの振る舞いを書き出すファイルを閉じる処理を行なっている。

【００７２】このようにして、埋め込み処理がなされた後、図２のステップＳ３に進み、システムを実行する。システムを実行すると、ステップＳ２で埋め込んだプリント文により、ファイル(ＯＯ＿ＴＯＯＬ＿ＦＩＬＥ＝"ＯＯ＿ｔｏｏｌ．ｆｉｌｅ")にオブジェクトの振る舞いが書き出され、時系列に保存される。その保存された内容を図１５に示す。すなわち、図１５は、図１２乃至図１４のプログラムをコンパイルし、システムを実行した結果書き出されたオブジェクトの振る舞いに関する情報を含んでいるファイルの内容である。

【００７３】図１５には、６つのオブジェクトの振る舞いが書き出されており、Ｐ１'およびＰ２'はオブジェクトの生成で、図１２乃至図１４の(Ｑ４)および(Ｑ５)にそれぞれ対応している。また、Ｐ３'およびＰ４'はメソッドの呼び出しであり、図１２乃至図１４の(Ｑ６)および(Ｑ２)にそれぞれ対応している。また、Ｐ５'およびＰ６'はオブジェクトの削除であり、図１２乃至図１４の(Ｑ７)および(Ｑ８)にそれぞれ対応している。

【００７４】このようにして、保存手段２によりオブジェクトの振る舞いがファイルに保存されると、図２のステップＳ４に進み、システムの実行を視覚的に見るために、出力手段３を起動する。これにより、出力手段３は、ファイルの内容を読み込み、ユーザの指定に従って視覚的にシステム内でのオブジェクトの振る舞いを表示する。ここで、ユーザの指定として、１）自動的に一定時間でオブジェクトの振る舞いを出力手段が進行していく、２）ユーザが逐次指示を行い、オブジェクトの振る舞いの表示を行なっていくなどの指定を選択して行なうことができる。

【００７５】実際に、図１５のファイル内容(履歴)に基づいて、オブジェクトの振る舞いを表示した出力手段３の様子の一例を図１６乃至図１８に示す。すなわち、図１６乃至図１８は、本実施例の開発支援装置が図１５のファイル内容を読み込んで、オブジェクトの振る舞いを表示している状態を示す図である。

【００７６】図１６乃至図１８を参照すると、まず、図１６(ａ)は出力手段起動時の画面であり、この時点では何も表示されていない。次に、出力手段３が起動されると、図１６(ｂ)のように、画面上にプログラムボックスが表示される。ここで、プログラムボックスが表示されるのは、出力手段３ではオブジェクトの振る舞いおよびオブジェクト間の振る舞いを表示する一方で、オブジェクトではなく関数からオブジェクトが作成されたり、メソッドが呼び出されたりする場合に、これらに対応した表示領域を確保するためである。なお、図１６(ｂ)の例では、便宜上、１つのプログラムボックスしか表示されていないが、ユーザの要求に応じて関数毎にあるいはモジュール毎にプログラムボックスを複数表示しても良い。

16

【００７７】次に、画面上には、図１６(ｃ)のように、図１５のＰ１'に対応する処理、すなわちｃｌａｓｓＡ型のオブジェクトａが作成される処理が表示される。すなわち、画面上にはクラス型Ａと変数名ａとがボックスの中に表示される。次いで、画面上には、図１６(ｄ)のように、図１５のＰ２'に対応するｃｌａｓｓＢ型のオブジェクトｂが作成される処理が表示される。すなわち、図１６(ｃ)と同様に、クラス型Ｂと変数名ｂとがボックスの中に表示される。

【００７８】次いで、画面上には、図１７(ｅ)のように、図１５のＰ３'に対応する処理が表示される。すなわち、Ｐ３'の処理は、メイン関数からのオブジェクトａの持つメソッドの呼び出しであるので、プログラムボックスからｃｌａｓｓＡ型のオブジェクトａボックスに矢印が表示される。ここで、オペレータが矢印を選択することにより、どのようなデータが渡されたかなどのメソッドに関する情報を表示することが可能である。次いで、画面上には、図１７(ｆ)のように、図１５のＰ４'に対応する処理が表示される。すなわち、Ｐ４'の処理では、Ｐ３'と同様にしてメソッドｍｅｔｈｏｄＡからｃｌａｓｓＢ型のオブジェクトの持つメソッドが呼び出されているので、オブジェクトａからオブジェクトｂに対して矢印が表示される。

【００７９】メソッドの処理が終了すると、メソッドの呼び出しを示す矢印が消去されるので、画面上では、図１７(ｇ)のように、オブジェクトａからのメソッドを示す矢印が消去され、また、図１７(ｈ)のように、プログラムからオブジェクトａのメソッドを呼び出すことを示す矢印が順次消去される。

【００８０】次に、画面上では、図１５のＰ５'の処理でオブジェクトｂが削除されるので、画面上では、図１８(ｉ)のように、オブジェクトｂを示すオブジェクトｂボックスが消去される。次いで、同様にして図１５のＰ６'の処理でオブジェクトａが削除されるので、画面上では、図１８(ｊ)のように、オブジェクトａボックスが削除され、一連のオブジェクトの振る舞いを示す処理が終了する。

【００８１】上述の例では、一連の画面(図１６(ａ)～図１８(ｊ))によるオブジェクトの振る舞いを表示したが、この処理はリアルタイム処理ではなく、ファイルにオブジェクトの振る舞いがすべて保存されている。従って、処理の実行を後戻りすることも可能である。すなわち、図１６(ａ)～図１８(ｊ)に示した一連の画面を途中から元に戻したり、また、図１８(ｊ)から逆に始めて図１６(ａ)に戻すことも可能である。

【００８２】

【発明の効果】以上に説明したように、請求項１乃至請

求項９記載の発明では、オブジェクト指向システムの開発環境において、オブジェクト指向システムを実際に起動した際に、オブジェクト間でのメッセージの授受やオブジェクトの生成や削除などのオブジェクトの振る舞いを認識するオブジェクト挙動認識手段と、オブジェクト挙動認識手段によって認識されたオブジェクトの振る舞いをファイルに保存する保存手段と、前記保存手段によってファイルに保存されたオブジェクトの振る舞いを読み取り出力する出力手段とを備えているので、オブジェクト指向システムを実際に動作させたときのシステムの挙動、すなわち、オブジェクトの振る舞いなどをユーザが容易に理解することができる。

【００８３】特に、請求項３記載の発明によれば、前記オブジェクト挙動認識手段は、特定のオブジェクトに注目し、該特定のオブジェクトに関する振る舞いのみを認識するので、必要なデータのみを集中して取り扱うことができ、不必要なデータを省くことができる。

【００８４】また、請求項６記載の発明では、さらに、前記オブジェクト挙動認識手段，前記データ認識手段を、オブジェクト指向システムの実行前にファイルに埋め込む第２の埋込手段を有しており、第２の埋込手段は、ソースプログラムを解析することによって、オブジェクトの生成や削除，メッセージの送信などの振る舞いの箇所を自動認識してオブジェクト挙動認識手段を埋め込むことができるので、ユーザは、オブジェクト挙動認識手段をこのオブジェクト指向システムに自ら設定する必要がなく、ユーザの作業効率を著しく向上させることができる。

【００８５】また、請求項７記載の発明では、前記出力手段は、前記ファイルに保存された内容を視覚化して表示するようになっており、例えば、システムの振る舞いをユーザに提示する場合には、オブジェクトを四角や円などで表示したり、また、メッセージの授受(やりとり)をユーザに提示する場合には、オブジェクト間にリンクを表示するなどによって、ユーザは視覚的に容易にシステムの振る舞いを把握できる。

【００８６】また、請求項８記載の発明では、前記出力手段は、前記ファイルに保存された内容を視覚化して表示する際に、ユーザの要求に応じて、システムの稼働に合わせて、逐次、システムの稼働状況を表示するので、時間の経過によるシステムの振る舞いをユーザに提示したり、また、時間をさかのぼってシステムの振る舞いをユーザに提示すること(すなわち、時間をさかのぼってシステムの振る舞いをたどること)も可能となる。

【００８７】また、請求項９記載の発明によれば、オブジェクトの振る舞いをシステムの実行と同時に視覚化して表示するので、ユーザは、リアルタイムに容易にシス＊テムの振る舞いを理解することができる。

【図面の簡単な説明】

【図１】本発明に係るオブジェクト指向システムの開発支援装置の一実施例の構成図である。

【図２】図１の開発支援装置の処理動作の一例を示すフローチャートである。

【図３】図１の開発支援装置の処理動作の他の例を示すフローチャートである。

【図４】データ型表の一例を示す図である。

【図５】クラス表の一例を示す図である。

【図６】変数表の一例を示す図である。

【図７】オブジェクト生成に対応したデバッグ用プログラムコードの一例を示す図である。

【図８】オブジェクト削除に対応したデバッグ用プログラムコードの一例を示す図である。

【図９】オブジェクトのもつメソッドの呼び出しに対応したデバッグ用プログラムコードの一例を示す図である。

【図１０】オブジェクトのもつ属性への値の代入に対応したデバッグ用プログラムコードの一例を示す図である。

【図１１】ソースプログラム(サンプルプログラム)の一例を示す図である。

【図１２】図１１のソースプログラムにデバッグ用プログラムコードが埋め込まれた状態を示す図である。

【図１３】図１１のソースプログラムにデバッグ用プログラムコードが埋め込まれた状態を示す図である。

【図１４】図１１のソースプログラムにデバッグ用プログラムコードが埋め込まれた状態を示す図である。

【図１５】図１２乃至図１４のプログラムをコンパイルし、システムを実行した結果書き出されたオブジェクトの振る舞いに関する情報を含んでいるファイルの内容を示す図である。

【図１６】図１５のファイル内容を読み込んで、オブジェクトの振る舞いを表示している状態を示す図である。

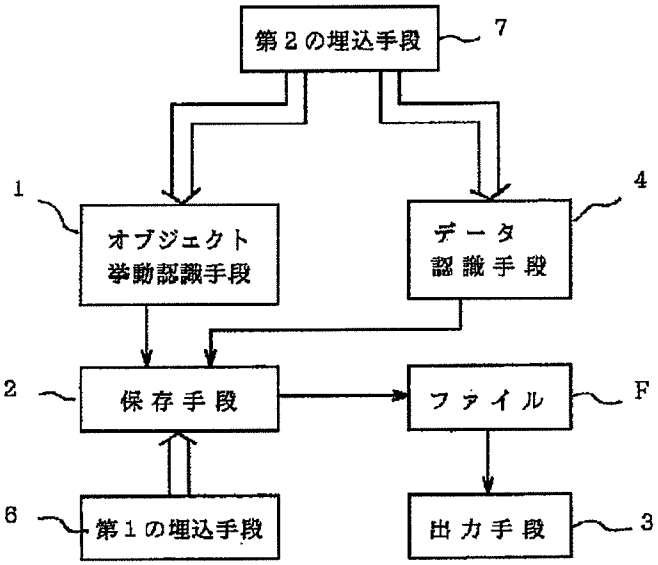
【図１７】図１５のファイル内容を読み込んで、オブジェクトの振る舞いを表示している状態を示す図である。

【図１８】図１５のファイル内容を読み込んで、オブジェクトの振る舞いを表示している状態を示す図である。

【符号の説明】

1	オブジェクト挙動認識手段
2	保存手段
3	出力手段
4	データ認識手段
6	第１の埋込手段
7	第２の埋込手段
F	ファイル

【図１】



【図４】

int
float
double
char
classA
classB

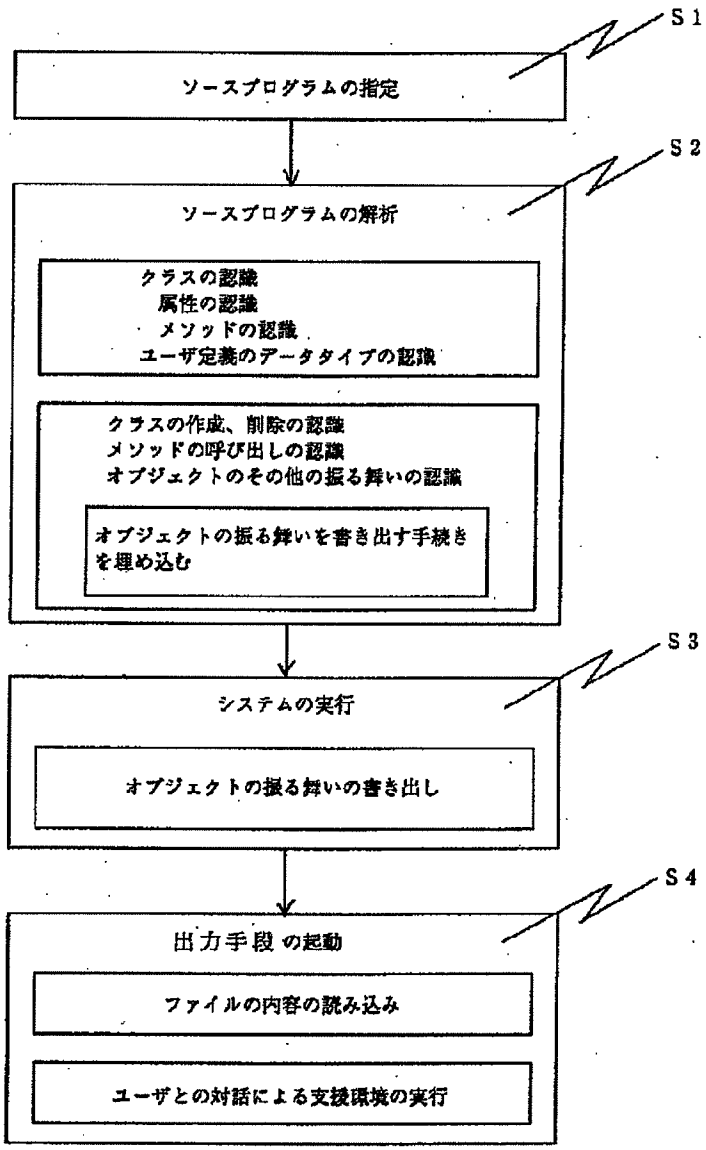
【図６】

関数名・メソッド名	変数名	変数型
main	i	int
	j	int
	a	classA
	b	classB

【図５】

クラス名	属性名	属性型	メソッド名	返り値型
classA	ptr	classB	methodA	int
classB			methodB	int

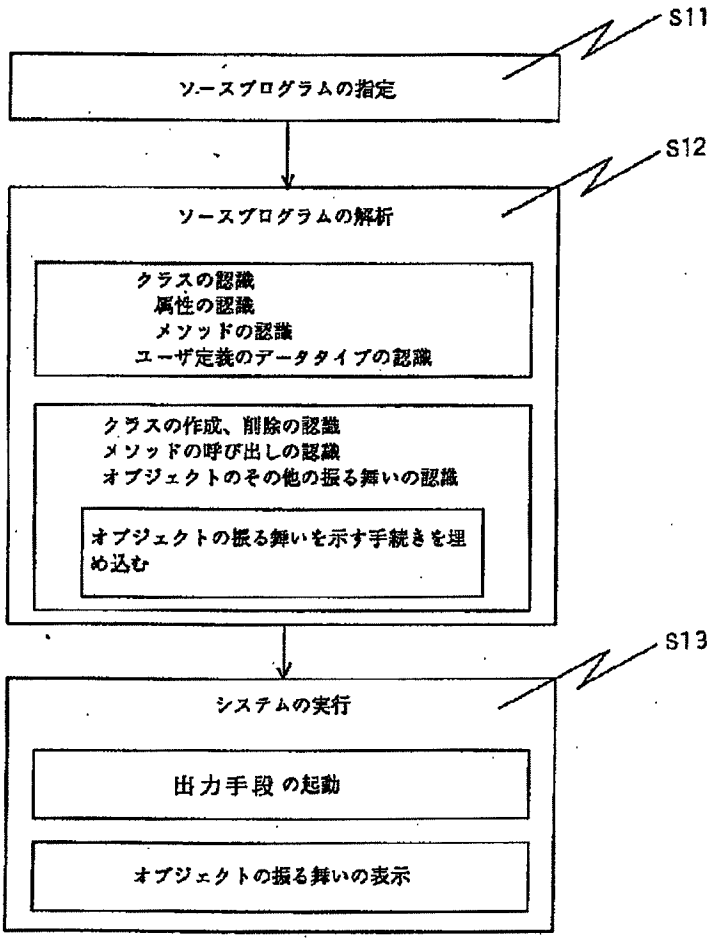
【図２】



【図７】

```
#ifdef OO_TOOL
fprintf(fp, "P%d :", ++count);
fprintf(fp, "\tCREATE : %s\n", クラス名);
fprintf(fp, "\tVARIABLE : %s\n", 変数名);
fprintf(fp, "\tWHERE : %s\n", 生成箇所);
fprintf(fp, "\tFUNCTION : %s\n", 関数名またはメソッド名);
#endif
```

【図３】



【図８】

```
#ifdef OO_TOOL
fprintf(fp, "P%d :", ++count);
fprintf(fp, "\tDELETE : %s\n", クラス名);
fprintf(fp, "\tVARIABLE : %s\n", 変数名);
fprintf(fp, "\tWHERE : %s\n", 削除箇所);
fprintf(fp, "\tFUNCTION : %s\n", 関数名またはメソッド名);
#endif
```

【図９】

```
#ifdef OO_TOOL
fprintf(fp, "P%d :", ++count);
fprintf(fp, "\tMETHOD : %s\n", メソッド名);
fprintf(fp, "\tFROM : %s\n", 関数名またはクラス名);
fprintf(fp, "\tFUNCTION : %s\n", 関数名またはメソッド名);
fprintf(fp, "\tTO : %s\n", 呼び出し先クラス名);
fprintf(fp, "\tARGC : %d\n", 引き数の数);
fprintf(fp, "\tARG : %d : %d\n", 引き数);
#endif
```

【図１０】

```
#ifdef OO_TOOL
fprintf(fp, "P%d :", ++count);
fprintf(fp, "\tATTRIBUTE : %s\n", 属性名);
fprintf(fp, "\tWHERE : %s\n", 関数名またはクラス名);
fprintf(fp, "\tFUNCTION : %s\n", 関数名またはメソッド名);
fprintf(fp, "\tTO : %s\n", 呼び出し先クラス名);
fprintf(fp, "\tVARIABLE : %d\n", 変数名);
#endif
```

【図１４】

```
    #ifdef OO_TOOL
    fprintf(fp, "P%d :", ++count);
    fprintf(fp, "\tDELETE : %s\n", "classA");
    fprintf(fp, "\tVARIABLE : %s\n", "a");          Q7
    fprintf(fp, "\tWHERE : %s\n", "FUNCTION");
    fprintf(fp, "\tFUNCTION : %s\n", "main");
    #endif
    delete a;

    #ifdef OO_TOOL
    fprintf(fp, "P%d :", ++count);
    fprintf(fp, "\tDELETE : %s\n", "classB");
    fprintf(fp, "\tVARIABLE : %s\n", "b");          Q8
    fprintf(fp, "\tWHERE : %s\n", "FUNCTION");
    fprintf(fp, "\tFUNCTION : %s\n", "main");
    #endif
    delete b;

    #ifdef OO_TOOL                                  Q9
    fclose(fp);
    #endif
}
```

【図１１】

ファイル名：test.cc

```
#include <stdio.h>

class classB{
public:
    int methodB(char c);
};

class classA{
public:
    classB *ptr;
    int methodA(int a, int b);
};

int classA::methodA(int a, int b){
    ptr->methodB('a'); ―――――――――― P1
}

main(){
    int i = 10;
    int j = 100;

    classA *a = new classA; ―――――――― P2
    classB *b = new classB; ―――――――― P3

    a->ptr = b;
    a->methodA(i, j); ―――――――――――― P4

    delete a; ――――――――――――――― P5
    delete b; ――――――――――――――― P6
}
```

【図１５】

```
P1' :
    CREATE : classA
    VARIABLE : a
    WHERE : FUNCTION
    FUNCTION : main

P2' :
    CREATE : classB
    VARIABLE : b
    WHERE : FUNCTION
    FUNCTION : main

P3' :
    METHOD : methodA
    FROM : FUNCTION
    FUNCTION : main
    TO : classA
    ARGC : 2
    ARG 1 : 10
    ARG 2 : 100

P4' :
    METHOD : methodB
    FROM : classA
    FUNCTION : methodA
    TO : classB
    ARGC : 1
    ARG 1 : a

P5' :
    DELETE : classA
    VARIABLE : a
    WHERE : FUNCTION
    FUNCTION : main

P6' :
    DELETE : classB
    VARIABLE : b
    WHERE : FUNCTION
    FUNCTION : main
```

【図１２】

```
#define OO_TOOL
#ifdef OO_TOOL
#indlude <stdio.h>
#define OO_TOOL_FILE "oo_tool.file"
FILE *fp;
int count = 0;
#endif

#include <stdio.h>

class classB{
public:
    int methodB(char c);
};


class classA{
public:
    classB *ptr;
    int methodA(int a, int b);
};

int classA::methodA(int a, int b){
    #ifdef OO_TOOL
    fprintf(fp, "P%d :", ++count);
    fprintf(fp, "\tMETHOD : %s\n", "methodB");
    fprintf(fp, "\tFROM : %s\n", "classA");
    fprintf(fp, "\tFUNCTION : %s\n", "methodA");
    fprintf(fp, "\tTO : %s\n", "classB");
    fprintf(fp, "\tARGC : %d\n", 1);
    fprintf(fp, "\tARG %d : %c\n", 1, 'a');
    #endif
    ptr->methodB('a');
}
```

Q1

Q2

【図１３】

```
main(){
    #ifdef OO_TOOL
    fp = fopen(OO_TOOL_FILE, "w");
    #endif

    int i = 10;
    int j = 100;

    #ifdef OO_TOOL
    fprintf(fp, "P%d :", ++count);
    fprintf(fp, "\tCREATE : %s\n", "classA");
    fprintf(fp, "\tVARIABLE : %s\n", "a");
    fprintf(fp, "\tWHERE : %s\n", "FUNCTION");
    fprintf(fp, "\tFUNCTION : %s\n", "main");
    #endif
    classA *a = new classA;

    #ifdef OO_TOOL
    fprintf(fp, "P%d :", ++count);
    fprintf(fp, "\tCREATE : %s\n", "classB");
    fprintf(fp, "\tVARIABLE : %s\n", "b");
    fprintf(fp, "\tWHERE : %s\n", "FUNCTION");
    fprintf(fp, "\tFUNCTION : %s\n", "main");
    #endif
    classB *b = new classB;

    a->ptr = b;
    #ifdef OO_TOOL
    fprintf(fp, "P%d :", ++count);
    fprintf(fp, "\tMETHOD : %s\n", "methodA");
    fprintf(fp, "\tFROM : %s\n", "FUNCTION");
    fprintf(fp, "\tFUNCTION : %s\n", "main");
    fprintf(fp, "\tTO : %s\n", "classA");
    fprintf(fp, "\tARGC : %d\n", 2);
    fprintf(fp, "\tARG %d : %d\n", 1, i);
    fprintf(fp, "\tARG %d : %d\n", 2, j);
    #endif
    a->methodA(i, j);
```

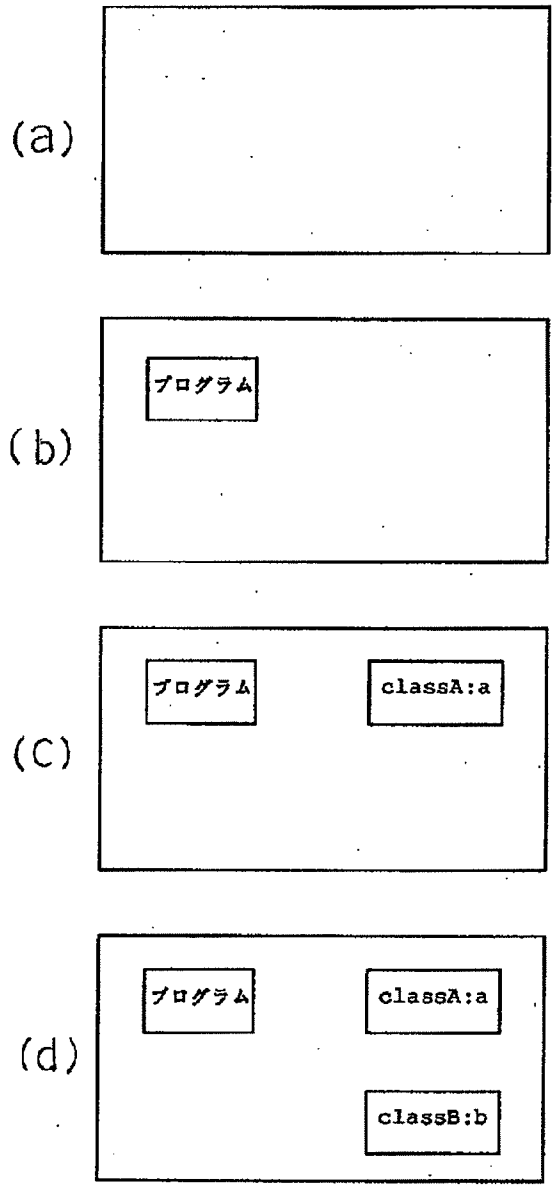
Q3

Q4

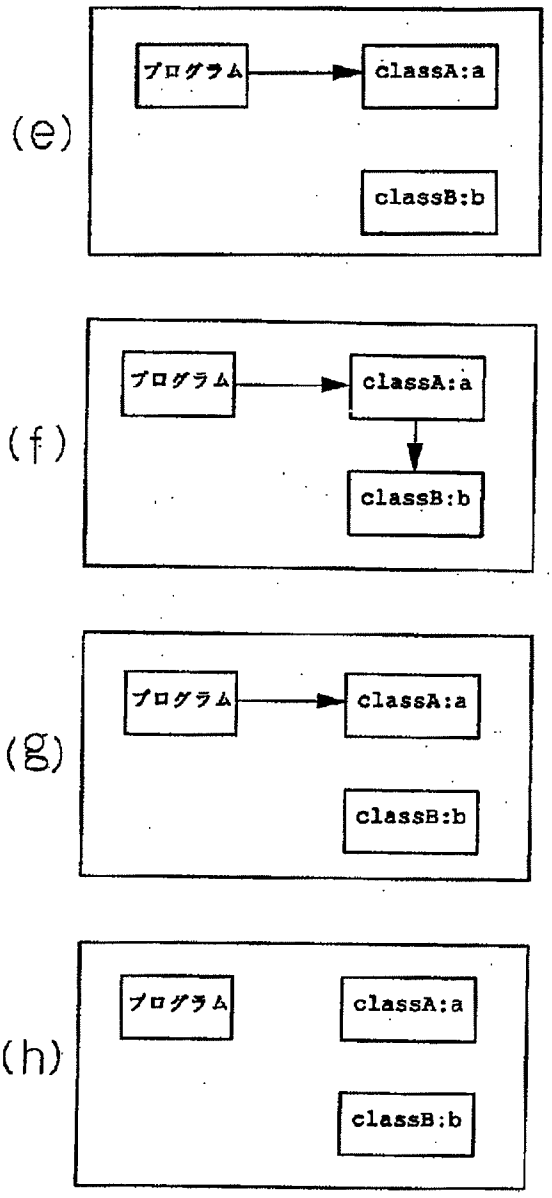
Q5

Q6

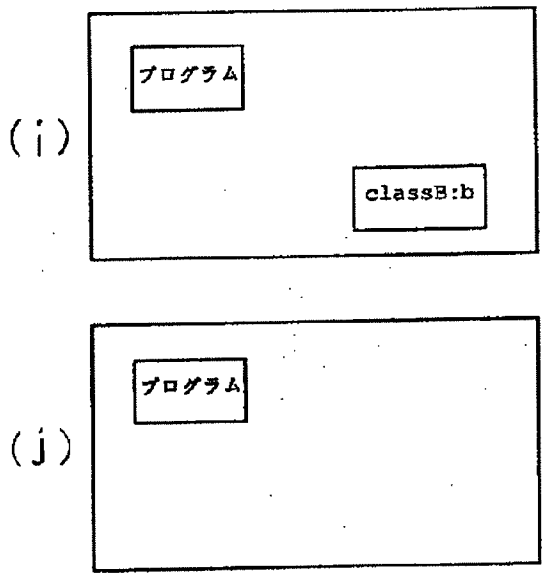
【図１６】



【図１７】


(e)
プログラム
classA:a
classB:b
(f)
プログラム
classA:a
classB:b
(g)
プログラム
classA:a
classB:b
(h)
プログラム
classA:a
classB:b

【図１８】


(i)
プログラム
classB:b
(j)
プログラム